Stanford Artificial Intelligence Laboratory November 1973

FINAL REPORT

RESEARCH IN MATHEMATICAL THEORY OF COMPUTATION

John McCarthy, Professor of Computer Science
Principal Investigator

supported by
National Aeronautics and Space Administration
under
Contract NSR 05-020-500

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

1. Introduction

This report summarizes research performed under NASA Contract NSR-05-020-500 in the period 1970 December 18 to 1973 April 1. The primary products of this work have already been published (Appendices A and B). Accomplishments under this contract can best be understood in terms of antecedent work.

2. Early Work

The idea that computer scientists should study computations themselves rather then just the notion of computability (i.e. recursion theory) was suggested in 1963 by McCarthy [1,2]. These early papers suggested that mathematical methods could be used to prove (or disprove) the following properties of programs:

1. a program is correct,
2. a program terminates,
3. two programs are equivalent,
4. a translation procedure between two languages is correct, (i.e. it preserves the meaning of a program),
5. optimized programs are equivalent to the original,
6. one program uses less resources than another and is therefore more efficient.

These are simply technical descriptions of a programer's day to day problems. The notion of correctness of a program is just - how do we know that a particular program solves the problem that it was intended to. The usual way of putting it is: "Does my program have bugs in it". A correct mathematical description of what this means is a central problem in MTC and is a genuine first step in any attempt to mechanize the debugging of programs. The equivalence of programs is similar in that until there are clear ways of describing what a program does, saying that they "do" the same thing is impossible. These technical problems are now well enough understood so that serious attempts to apply the results to "real" programs are beginning. Attempts to formalize these questions have proceeded along several lines simultaneously. In [4,6] McCarthy and Mansfield discussed new languages for expressing these notions were considered. [4] considered a first order logic which contained an "undefined" truth-value. This was one way of explaining what was meant by computations which didn't terminate. [5] used a traditional first order logic to describe a subset of ALGOL.

In [3] McCarthy proposed that computers themselves might be used to check the correctness of proofs in formal systems, and was the first to actually construct a program to carry this out. This suggests that one could check or possibly look for solutions to the above problems (in the form of proofs in some formal system). As a result a series of proof checkers has been built. The first is reported in [7].

In 1966 Floyd [8] published his now well known method of assigning assertions to the paths in a flowchart, in order to find verification conditions the truth of which guarantee the "correctness" of the original program.

McCarthy, Painter and Kaplan [9,10,11,12,13,14] used the ideas in [4,8] to prove:

1) the correctness of a compiler for arithmetic expressions,
2) the correctness of several compilers for algol-like programs,
3) the equivalence of some algorithms.

Kaplan also gave some completeness results for a formal system which talks about assignment statements [10], and discussed the equivalence of programs [13,14]. During this time another proof checker was written by W. Weiher [15].

In a series of articles Z. Manna extended and expanded Floyd's original ideas. With A. Pneulli [16,17] he discussed the relationship between the termination, correctness and equivalence of recursively defined functions and the satisfiability (or unsatisfiability) of certain first order formulas. In [17] they work out an example using the 91 function. In [18] Manna extended his ideas to non-dererministic programs. E. Ashcroft and he did a similar thing for parallel programs in [21].

P. Hayes [18] again attacked the problem of a three-valued predicate logic, this time with a machine implementation in mind. This coincided with a paper of Manna and McCarthy [19], which used this logic.

About this time (1969) several important developments occurred which allowed the above questions to be reexamined from different points of view.

1. In [22] Z. Manna showed how to formulate the notion of partial correctness in second logic.
2. C. A. R. Hoare [24] published a paper describing a new formalism for expressing the meanings of programs in terms of input/output relations.
3. S. Igarashi [23] gave an axiomatic description of an ALGOL-like language.
4. D. Scott suggested using the typed lambda calculus for studying MTC and first described IN 1970 a mathematical model of Church's lambda calculus.

These together with McCarthy's axiomatic approach now represent the most important directions in MTC research. They express different points of view towards the meanings (or semantics) of programs.

3. Research Performed under this Contract

By the beginning of the contract period (December 1970), Mathematical Theory of Computation had become quite a lively field. Research performed under this contract cannot be cleanly separated from other concurrent work at Stanford (mainly by Edward Ashcroft, Ashok Chandra, Robert Floyd, Shigeru Igarashi, and Ralph London), or from work elsewhere. Instead, we simply summarize recent progress, with a note that the following persons received at least part of their support from the subject contract: Jack Buchanan, John McCarthy, Zohar Manna, Robin Milner, and Richard Weyhrauch.

Manna (following Floyd) describes the effects of a program by showing what kinds of relations must hold among the values of the program variables at different points in the execution of the program. In particular between the input and the output. In [31] Floyd suggests an interactive system for designing correct programs. These ideas are systematized and expanded by Manna in [34]. He and Ashcroft show how to remove GOTO statements from programs and replace them by WHILE statements in [33].

Hoare shows how properties (including the meaning) of a program can be expressed as rules of inference in his formal system and how these rules can be used to generate the relations described by Floyd and Manna. This puts their approach in a formal setting suitable for treatment on a computer. Work on this formal system is at present being aggressively pursued. Igarashi, London, and Luckham have increased the scope of the original rules and have programed a system called VCG (for verification condition generator) which takes PASCAL programs together with assertions assigned to loops in the program and uses the Hoare rules to automatically generate verification conditions the proof of which guarantee the correctness of the original program. These sentences are then given to a resolution theorem prover[26] which tries to prove them. There is also a project started by Suzuki under the direction of Luckham to develop programs to take account of particular properties of arithmetic and arrays when trying to prove the verification conditions. London also produced an informal proof of two lisp compiler[35] Igarashi's formal system [23] differs from Hoare's in that the rules of inference act directly on the programs themselves rather than properties of such programs.

Scott's work assumes that the most suitable meaning for a program is the function which it computes and essentially ignores how that computation proceeds. The other approaches are more intentional in that:

1) they may not necessarily mention that function explicitly although it might appear implicitly.
2) they can (and do) consider notions of meaning that are stronger than Scott's.

For example programs might have to have "similar" computation sequences before considering them equivalent[25].

A computer program (LCF for "logic for computable functions") has been implemented by Milner[26]. This logic uses the typed lambda calculus to defines the semantics of programs. Exactly how to do this was worked out by Weyhrauch and Milner[28,29,30]. In conjunction Newey worked on the axiomatization of arithmetic, finite sets, and lists in the LCF environment.

This work is still continuing. In addition Milner and Weyhrauch worked with Scott on an axiomatization of the type free lambda calculus. Much of this work was informally summarized in [32].

McCarthy attempts to give an axiomatic treatment to a programming language by describing its abstract syntax in first order logic and stating properties of the programming language directly as axioms. This approach has prompted Weyhrauch to begin the design of a new first order logic proof checker based on natural deduction. This proof checker is expected to incorporate the more interesting features of LCF and will draw heavily on the knowledge gained from using LCF to attempt to make the new first order proof checker a viable tool for use in proving properties of programs.

This work is all being brought together by projects that are still to a large extent unfinished. They include

1) a new version of LCF including a facility to search for proofs automatically.
2) the description of the language PASCAL in terms of both LCF and in first order logic (in the style of McCarthy) in order to have a realistic comparison between these approaches and that of Floyd, Hoare, et al.
3) a continuation of Newey's work.
4) the discussion of LISP semantics in LCF and an attempt to prove the correctness of the London compilers in a formal way. This is also being done by Newey.
5) the design of both special purpose and domain independent proving procedures specifically with program correctness in mind.
6) the design of languages for describing such proof procedures
7) the embedding of these ideas in the new first order checker.

In addition to the work described above, Ashcroft, Manna, and Pneuli[36], and Chandra and Manna [37] have published results related to program schemas.

Some of these references appeared both as A.I. memos and were later published in journals. In such cases both references appear in the bibliography.

REFERENCES

[1] McCarthy, John, "A Basis for a Mathematical Theory of Computation," in Biaffort, P., and Herschberg, D., (eds.), COMPUTER PROGRAMMING AND FORMAL SYSTEMS. Amsterdam: North-Holland, 1963.

[2] McCarthy, John, "Towards a Mathematical Theory of Computation," in PROC. IFIP CONGRESS 62. Amsterdam: North-Holland, 1963.

[3] McCarthy, John, "Checking Mathematical Proofs by Computer," in Proc. Symp. on Recursive Function Theory(1961). American Mathematical Society, 1962.

[4] McCarthy, John, PREDICATE CALCULUS WITH "UNDEFINED" AS A TRUTH-VALUE. AIM-1, March, 1963.

[5] McCarthy, John, A FORMAL DESCRIPTION OF A SUBSET OF ALGOL. AIM-24, September, 1964.

----------, in Steele, T., (ed.), FORMAL LANGUAGE DESCRIPTION LANGUAGES. Amsterdam: North Holland, 1966.

[6] Mansfield, R., A FORMAL SYSTEM OF COMPUTATION. AIM-25, September 1964.

[7] McCarthy, John, A PROOF-CHECKER FOR PREDICATE CALCULUS. AIM-27, March, 1965.

[8] Floyd, R. W., "Assigning Meanings to programs," in Proc. Symp. in vol. 19, American Mathematical Society, 19-32 1967.

[9] McCarthy, John, and Painter, J., CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS. AIM-40, April, 1966.

----------, in MATHEMATICAL ASPECTS OF COMPUTER SCIENCE. New York: American Mathematical Society Proc. Symposia in Applied Mathematics, 1967. [10] Painter, J., SEMANTIC CORRECTNESS OF A COMPILER FOR AN ALGOL-LIKE LANGUAGE. AIM-44, March, 1967.

[11] Kaplan, D., SOME COMPLETENESS RESULTS IN THE MATHEMATICAL THEORY OF COMPUTATION, AIM-45, October, 1966.

----------, in ACM JOURNAL, January 1968.

[12] Kaplan, D., CORRECTNESS OF A COMPILER FOR ALGOL-LIKE PROGRAMS. AIM-48, July, 1967

[13] Kaplan, D., A FORMAL THEORY CONCERNING THE EQUIVALENCE OF ALGORITHMS. AIM-59, May, 1968.

[14] Kaplan, D., THE FORMAL THEORETIC ANALYSIS OF STRONG EQUIVALENCE FOR ELEMENTAL PROGRAMS. AIM-60, June, 1968.

[15] Weiher, William, THE PDP-6 PROOF CHECKER. AIM-53, June, 1967.

[17] Manna, Zohar, and Pneuli, Amir, FORMALIZATION OF PROPERTIES OF RECURSIVELY DEFINED FUNCTIONS. AIM-82, March, 1969.

----------, in ACM JOURNAL, Vol. 17, No. 3. July, 1970.

[18] Hayes, Patrick J., A MACHINE-ORIENTED FORMULATION OF THE EXTENDED FUNCTIONAL CALCULUS. AIM-86, June, 1969.

[19] Manna, Zohar, and McCarthy, John, PROPERTIES OF PROGRAMS AND PARTIAL FUNCTION LOGIC. AIM-100, October, 1969.

----------,in Meltzer, B. and Michie, D. (eds.), MACHINE INTELLIGENCE 5. Edinburgh: Edinburgh University Press, 1970.

[20] Manna, Zohar, THE CORRECTNESS OF NON-DETERMINISTIC PROGRAMS. AIM-95, August, 1969.

----------, in ARTIFICIAL INTELLIGENCE JOURNAL, Vol. 1, No. 1, 1970.

[21] Ashcroft, Edward, and Manna, Zohar, FORMALIZATION OF PROPERTIES OF PARALLEL PROGRAMS. AIM-110, February, 1970.

----------, in MACHINE INTELLIGENCE 6. Edinburgh: Edinburgh University Press, 1971.

[22] Manna, Zohar, SECOND-ORDER MATHEMATICAL THEORY OF COMPUTATION. AIM-111, March, 1970.

----------, in PROC. ACM SYMPOSIUM ON THEORY OF COMPUTING. May, 1970.

[23] Igarashi, Shigeru, SEMANTICS OF ALGOL-LIKE STATEMENTS. AIM-129, June, 1970.

[24] Hoare, C.A.R., "An Axiomatic Basis for Computer Programming", in Comm. ACM 12, No. 10, pp.576-580, 1969.

[25] Milner, Robin, AN ALGEBRAIC DEFINITION OF SIMULATION BETWEEN PROGRAMS. AIM-142, February, 1971.

----------, in PROC. 2IJCAI. British Computer Society, 1971.

[26] Allen, John and Luckham, David, AN INTERACTIVE THEOREM-PROVING PROGRAM. AIM-103, October, 1971.

[27] Milner, Robin, LOGIC FOR COMPUTABLE FUNCTIONS; DESCRIPTION OF A MACHINE IMPLEMENTATION. AIM-169, May 1972.

[28] Milner, Robin, IMPLEMENTATION AND APPLICATIONS OF SCOTT'S LOGIC FOR COMPUTABLE FUNCTIONS. Proc. Conf. on Proving Assertions about Programs. New Mexico State University, 1972.

[29] Milner, Robin and Weyhrauch, Richard, PROVING COMPILER CORRECTNESS IN A MECHANISED LOGIC, Machine Intelligence 7, Edinburgh University Press, 1972. [30] Weyhrauch, Richard and Milner, Robin, PROGRAM SEMANTICS AND CORRECTNESS IN A MECHANIZED LOGIC, Proc. USA-Japan Computer Conference, Tokyo, 1972.

[31] Floyd, Robert W., TOWARD INTERACTIVE DESIGN OF CORRECT PROGRAMS. AIM-150, September 1971.

----------, in PROC. IFIP CONGRESS. 1971.

[32] Manna, Zohar, Ness, Stephen, and Vuillemin, Jean, INDUCTIVE METHODS FOR PROVING PROPERTIES OF PROGRAMS. AIM-154, November, 1971.

----------, in ACM SIGPLAN NOTICES, Vol. 7, No. 4. 1972.

[33] Ashcroft, Edward, and Manna, Zohar, THE TRANSLATION OF 'GO-TO' PROGRAMS TO 'WHILE' PROGRAMS. AIM-138, January, 1971.

----------, in PROC. IFIP CONGRESS. 1971.

[34] Manna, Zohar, MATHEMATICAL THEORY OF PARTIAL CORRECTNESS. AIM-139, January, 1971.

----------, in J. COMP. AND SYS. SCI., June 1971.

[35] London, Ralph L., CORRECTNESS OF TWO COMPILERS FOR A LISP SUBSET. AIM-151, October, 1971.

[36] Ashcroft, Edward, Manna, Zohar, and Pneuli, Amir, DECIDABLE PROPERTIES OF MONADIC FUNCTIONAL SCHEMAS, AIM-148, July, 1971.

[37] Chandra, Ashok, and Manna, Zohar, PROGRAM SCHEMAS WITH EQUALITY. AIM-158, December, 1971.

Appendix A

PUBLICATIONS

Published Articles and books by members of the research staff who received at least partial support from the subject contract are listed below.

1. E. Ashcroft, Z. Manna, and A. Pnueli, "Decidable Properties of Monodic Functional Schemas", J. ACM, July 1973.

2. J. M. Cadiou and Z. Manna, "Recursive Definitions of Partial Functions and their Computations", ACM SIGPLAN Notices, Vol. 7, No. 1, January 1972.

3. Shmuel Katz and Zohar Manna, "A Heuristic Approach to Program Verification", Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, August 1973.

4. Z. Manna (with R. Waldinger), "Toward Automatic Program Synthesis", Comm. ACM, March 1971.

5. Z. Manna, "Mathematical Theory of Partial Correctness", J. Comp. & Sys. Sci., June 1971.

6. Z. Manna, S. Ness, and J. Vuillemin, "Inductive Methods for Proving Properties of Programs", ACM SIGPLAN Notices, Vol. 7, No. 4, January 1972.

7. Z. Manna and J. Vuillemin, "Fixpoint Approach to the Theory of Computation", Comm. ACM, July 1972.

8. Zohar Manna, "Program Schemas", in Currents in the Theory of Computing (A. V. Aho, Ed.), Prentice-Hall, Englewood Cliffs, N. J., 1973.

9. Zohar Manna, Stephen Ness, Jean Vuillemin, "Inductive Methods for Proving Properties of Programs", Comm. ACM, August 1973.

10. Zohar Manna, "Automatic Programming", Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, August 1973.

11. Zohar Manna, INTRODUCTION TO MATHEMATICAL THEORY OF COMPUTATION, McGraw-Hill, New York, 1974.

12. R. Milner, "An Algebraic Definition of Simulation between Programs", Proc. 2IJCAI, Brit. Comp. Soc., Sept. 1971.

13. R. Milner, "Implementation and Application of Scott's Logic for Computable Functions", ACM SIGPLAN NOTICES, Vol. 7, No. 1, January 1972.

14. Robin Milner and Richard Weyhrauch, "Proving Compiler Correctness in a Mechanized Logic", Machine Intelligence 7, Edinburgh University Press, 1972.

15. Richard Weyhrauch and Robin Milner, "Program Semantics and Correctness in a Mechanized Logic", Proc. USA-Japan Computer Conference, Tokyo, 1972.

Appendix B

ARTIFICIAL INTELLIGENCE MEMOS

Abstracts of research reports published by staff members who were supported by the subject contract are listed here.

AIM-138, Edward Ashcroft and Zohar Manna, THE TRANSLATION OF 'GO-TO' PROGRAMS TO 'WHILE' PROGRAMS, January 1971, 28 pages.

In this paper we show that every flowchart program can be written without 'go-to' statements by using 'while' statements. The main idea is to introduce new variables to preserve the values of certain variables at particular points in the program; or alternatively, to introduce special boolean variables to keep information about the course of the computation.

The 'while' programs produced yield the same final results as the original flowchart program but need not perform computations in exactly the same way. However, the new programs preserve the 'topology' of the original flowchart program, and are of the same order of efficiency.

We also show that this cannot be done in general without adding variables.

AIM-139, Zohar Manna, MATHEMATICAL THEORY OF PARTIAL CORRECTNESS, January 1971, 24 pages.

In this work we show that it is possible to express most properties regularly observed in algorithms in terms of 'partial correctness' (i.e., the property that the final results of the algorithm, if any, satisfy some given input-output relation).

This result is of special interest since 'partial correctness' has already been formulated in predicate calculus and in partial function logic for many classes of algorithms.

AIM-142, Robin Milner, AN ALGEBRAIC DEFINITION OF SIMULATION BETWEEN PROGRAMS, February 1971, 21 pages.

A simulation relation between programs is defined which is quasi-ordering. Mutual simulation is then an equivalence relation, and by dividing out by it we abstract from a program such details as how the sequencing is controlled and how data is represented. The equivalence classes are approximations to the algorithms which are realized, or expressed, by their member programs.

A technique is given and illustrated for proving simulation and equivalence of programs; there is an analogy with Floyd's technique for proving correctness of programs. Finally, necessary and sufficient conditions for simulation are given.

AIM-148, Edward Ashcroft, Zohar Manna, and Amir Pneuli, DECIDABLE PROPERTIES OF MONADIC FUNCTIONAL SCHEMAS, July 1971, 10 pages.

We define a class of (monadic) functional schemas which properly include 'Ianov' flowchart schemas. We show that the termination, divergence and freedom problems for functional schemas are decidable. Although it is possible to translate a large class of non-free functional schemas into equivalent free functional schemas, we show that this cannot be done in general. We show also that the equivalence problem for free functional schemas is decidable. Most of the results are obtained from well-known results in Formal Languages and Automata Theory.

AIM-154, Zohar Manna, Stephen Ness and Jean Vuillemin, INDUCTIVE METHODS FOR PROVING PROPERTIES OF PROGRAMS, November 1971, 24 pages.

We have two main purposes in this paper. First, we clarify and extend known results about computation of recursive programs, emphasizing the difference between the theoretical and practical approaches. Secondly, we present and examine various known methods for proving properties of recursive programs. We discuss in detail two powerful inductive methods, computational induction and structural induction, illustrating their applications by various examples. We also briefly discuss some other related methods.

Our aim in this work is to introduce inductive methods to as wide a class of readers as possible and to demonstrate their power as practical techniques. We ask the forgiveness of our more theoretical-minded colleagues for our occasional choice of clarity over precision.

AIM-158, Ashok Chandra, Zohar Manna, PROGRAM SCHEMAS WITH EQUALITY, December 1971, 13 pages.

We discuss the class of program schemas augmented with equality tests, that is, tests of equality between terms.

In the first part of the paper we illustrate the "power" of equality tests. It turns out that the class of program schemas with equality is more powerful than the "maximal" classes of schemas suggested by other investigators.

In the second part of the paper, we discuss the decision problems of program schemas with equality. It is shown, for example, that while the decision problems normally considered for schemas (such as halting, divergence, equivalence, isomorphism and freedom) are decidable for Ianov schemas. They all become undecidable if general equality tests are added. We suggest, however, limited equality tests which can be added to certain subclasses of program schemas while preserving their solvable properties.

AIM-163, J.M. Cadiou, RECURSIVE DEFINITIONS OF PARTIAL FUNCTIONS AND THEIR COMPUTATIONS, April 1972, 160 pages.

A formal syntactic and semantic model is presented for 'recursive definitions' which are generalizations of those found in LISP, for example. Such recursive definitions can have two classes of fixpoints, the strong fixpoints and the weak fixpoints, and also possess a class of computed partial functions.

Relations between these classes are presented: fixpoints are shown to be extensions of computed functions. More precisely, strong fixpoints are shown to be extensions of computed functions when the computations may involve "call by name" substitutions; weak fixpoints are shown to be extensions of computed functions when the computation only involve "call by value" substitutions. The Church-Rosser property for recursive definitions with fixpoints also follows from these results.

Then conditions are given on the recursive definitions to ensure that they possess least fixpoints (of both classes), and computation rules are given for computing these two fixpoints: the "full" computation rule, which leads to the least weak fixpoint. A general class of computation rules, called 'safe innermost', also lead to the latter fixpoint. The "leftmost innermost" rule is a special case of those, for the LISP recursive definitions.

AIM-164, Zohar Manna and Jean Vuillemin, FIXPOINT APPROACH TO THE THEORY OF COMPUTATION, April 1972, 29 pages.

Following the fixpoint theory of Scott, we propose to define the semantics of computer programs in terms of the least fixpoints of recursive programs. This allows one not only to justify all existing verification techniques, but also to extend them to handle various properties of computer programs, including correctness, termination and equivalence, in a uniform manner.

AIM-169, Robin Milner, LOGIC FOR COMPUTABLE FUNCTIONS. DESCRIPTION OF A MACHINE IMPLEMENTATION, May 1972, 36 pages.

This paper is primarily a user's manual for LCF, a proof-checking program for a logic of computable functions proposed by Dana Scott in 1969, but unpublished by him. We use the name LCF also for the logic itself, which is presented at the start of the paper. The proof-checking program is designed to allow the user interactively to generate formal proofs about computable functions and functionals over a variety of domains, including those of interest to the computer scientist--for example, integers, lists and computer programs and their semantics. The user's task is alleviated by two features: a subgoaling facility and a powerful simplification mechanism. Applications include proofs of program correctness and in particular of compiler correctness; these applications are not discussed herein, but are illustrated in the papers referenced in the introduction.

AIM-184, Malcolm Newey, AXIOMS AND THEOREMS FOR INTEGERS, LISTS AND FINITE SETS IN LCF, January 1973, 53 pages.

LCF (Logic for Computable Functions) is being promoted as a formal language suitable for the discussion of various problems in the Mathematical Theory of Computation (MTC). To this end, several examples of MTC problems have been formalised and proofs have been exhibited using the LCF proof-checker. However, in these examples, there has been a certain amount of ad-hocery in the proofs; namely many mathematical theorems have been assumed without proof and no axiomatisation of the mathematical domains involved was given. This paper describes a suitable mathematical environment for future LCF experiments and its axiomatic basis. The environment developed deemed appropriate for such experiments, consists of a large body of theorems from the areas of integer arithmetic, list manipulation and finite set theory.

AIM-185, Ashok K.Chandra and Zohar Manna, ON THE POWER OF PROGRAMMING FEATURES, January 1973, 29 pages.

We consider the power of several programming features such as counters, pushdown stacks, queues, arrays, recursion and equality. In this study program schemas are used as the model for computation. The relations between the powers of these features is completely described by a comparison diagram.

AIM-186, Robin Milner, MODELS OF LCF, January 1973, 17 pages.

LCF is a deductive system for computable functions proposed by D. Scott in 1969 in an unpublished memorandum. The purpose of the present paper is to demonstrate the soundness of the system with respect to certain models, which are partially ordered domains of continuous functions. This demonstration was supplied by Scott in his memorandum; the present paper is merely intended to make this work more accessible.

AIM-210, Zohar Manna and Amir Pnueli, AXIOMATIC APPROACH TO TOTAL CORRECTNESS OF PROGRAMS, July 1973, 25 pp.

We present here an axiomatic approach which enables one to prove by formal methods that his program is "totally correct" (i.e., it terminates and is logically correct -- does what it is supposed to do). The approach is similar to Hoare's approach for proving that a program is "partially correct" (i.e., that whenever it terminates it produces correct results). Our extension to Hoare's method lies in the possibility of proving correctness and termination at once, and in the enlarged scope of properties that can be proved by it.